TD sur la généricité en Java

Question 1. (Soit la classe ci-dessous :

```
public class C {
   public static void f(ArrayList<Integer> a) { }
   public static void f(ArrayList<String> s)  { }
   public static void main(String[] args) {
     f(new ArrayList<Integer>()); f(new ArrayList<String>());
   }
}
```

La définition de classe compile-t-elle ? Si « oui » qu'obtient-on à l'exécution, si « non » dites pourquoi.

Question 2 : on considère la classe générique `Paire<T1, T2>` vue en cours.

a. Écrivez une méthode `equals` adaptée à cette classe. Votre méthode `equals` risque-t-elle de lever des exception ?

On considère les déclarations suivantes :

```
Paire<Integer, Integer> s1 = new Paire<>(1, 1);
Paire<Integer, Integer> s2 = new Paire<>(1, 1);
Paire<String, String> s3 = new Paire<>("a", "b");
```

b. Détaillez l'exécution de chacun des deux appels ci-dessous :

```
s1.equals(s2);
s1.equals(s3);
```

c. Obtient-on toujours `false` dès qu'on applique `equals` sur deux instantiations différentes de `Paire` ? Justifiez votre réponse.

Question 3 : L'API Java définit l'interface `Collection<E>` dont `SortedSet` hérite et liste des méthodes supplémentaires pour parcourir des « ensembles ordonnés», le parcours se faisant selon l'ordre sur leurs éléments. `TreeSet<E>` est une classe implémentant `SortedSet<E>`.

1. Pourquoi a-t-on l'en-tête `SortedSet<E>` et non pas `SortedSet<E extends Comparable<E>>` comme on aurait pu le penser, puisqu'on doit pouvoir comparer des éléments (il en est de même pour `TreeSet`) ? Deux réponses attendues.

2. On considère l'extrait ci-dessous de l'API de `TreeSet` :

`TreeSet``()`

Constructs a new, empty tree set, sorted according to the natural ordering of its elements.

`TreeSet(Collection< … > c)`

Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.

`TreeSet(Comparator< … > comparator)`

Constructs a new, empty tree set, sorted according to the specified comparator.

Indiquez par quoi remplacer les … dans les en-têtes des deux derniers constructeurs. Justifiez brièvement votre réponse et l'intérêt de votre définition du paramètre.

3. Pour chacune des trois instructions ci-dessous, indiquez si elle compile et si elle s'exécute correctement. Justifiez brièvement vos réponses. Les classes `Object`, `Rectangle` et `Cercle` n'implémentent pas `Comparable`.

```
SortedSet<Object> s = new TreeSet<Object>();
s.add(new Rectangle());
s.add(new Cercle());
```

Question 4 : *Justifiez votre réponse en une ligne.*

```
public class A {
  private void f()
    { System.out.println("A::f"); }
  public void h() { this.f(); }
  public void k() { this.f(); }
  public static void g() {
    { System.out.println("A::g"); }
}
```

```
public class B extends A {
  public void f()
    { System.out.println("B::f"); }
  public void h() { this.f(); }

  public static void g() {
    { System.out.println("B::g"); }
}
```

1. La définition de `f` dans `B` est-elle légale ? Est-ce une redéfinition du `f` de `A` ou une méthode indépendante ?

2. Même question si on échange `private` / `public` pour `f` dans `A` et `B` ?

3. Pour chaque appel (s'il compile), indiquez le résultat de l'impression obtenue :

```
A monA = new A();   B monB = new B();
monA.g();
monA.h();
monB.g();
monB.h();
monA = monB;
monA.f();
monA.g();
monA.h();
monA.k();
```

Question 5 : On considère la classe `Collections` qui définit des méthodes **statiques** pour les collections. Fournissez les types manquants dans les déclarations **en favorisant l'applicabilité des méthodes. L'usage de classes de collections non génériques est bien sûr interdit** !

```
A. boolean disjoint(List< ... > l1, List< ... > l2)
```

cette fonction teste si les listes `l1` et `l2` sont disjointes.

B. `<T> int binarySearch(List < ... > l, … k, Comparator< ... > c)`

recherche dichotomique de `k` dans `l`; `l` doit être triée selon l'ordre défini par `c`.

C. `<T> void fill(List< ... > l, ... e)`

remplace tous les éléments de `l` par `e`.

D. `< ... > void sort(List<T> l)`

trie `l` selon l'ordre naturel des éléments de la liste

Question 6 : Soit trois classes `A1`, `A2`, `A3` reliées en chaîne par héritage (`A1` étant la plus générale) et les fonctions de signture ci-dessous :

```
void f1(ArrayList<A2> arg);
void f2(ArrayList<?> arg);
void f3(ArrayList<? extends A2> arg);
<T extends A2> void f4(ArrayList<T> arg);
```

a) quel type statique d'`ArrayList` peut-on passer en argument à l'appel ?

	f1	f2	f3	f4
ArrayList<A1>	[redacted]			
ArrayList<A2>	[redacted]			
ArrayList<A3>				

b) dans le corps de la fonction, peut-on écrire `arg.add(new A2());`

f1	f2	f3	f4
[redacted]			

c) dans le corps de la fonction, peut-on écrire `A2 e = arg.get(0);`

f1	f2	f3	f4
[redacted]			

d) Nos fonctions doivent maintenant renvoyer un élément de `arg` plutôt que `void`.
Quel type de retour écrivez-vous ?

f1	f2	f3	f4
[redacted]			

Une case noircie correspond à une question dont la réponse est immédiate et donc sans intérêt.